# Refine Search

**Search Results -**

| Terms | Documents |
|---|---|
| 707/100 | 5128 |

**Database:** US Pre-Grant Publication Full-Text Database; US Patents Full-Text Database; US OCR Full-Text Database; EPO Abstracts Database; JPO Abstracts Database; Derwent World Patents Index; IBM Technical Disclosure Bulletins

**Search:**

Refine Search | Recall Text | Clear | Interrupt

## Search History

**DATE: Wednesday, September 22, 2004** Printable Copy Create Case

| Set Name side by side | Query | Hit Count | Set Name result set |
|---|---|---|---|
| *DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=OR* | | | |
| L23 | 707/100 | 5128 | L23 |
| L22 | l6 and object with view | 86 | L22 |
| L21 | l6 and table with view | 94 | L21 |
| L20 | l6 and view | 124 | L20 |
| L19 | 6134558.pn. | 2 | L19 |
| L18 | 6128621.pn. | 2 | L18 |
| L17 | 6112207.pn. | 2 | L17 |
| L16 | 6061690.pn. | 2 | L16 |
| L15 | 6112210.pn. | 2 | L15 |
| L14 | 707.clas. | 22475 | L14 |
| L13 | 707/103r | 1518 | L13 |
| L12 | 707/102 | 5076 | L12 |
| L11 | 707/10 | 9129 | L11 |

| | | | |
|---|---|---|---|
| L10 | 707/4 | 3937 | L10 |
| L9 | 707/3 | 7045 | L9 |
| L8 | 707/2 | 4205 | L8 |
| L7 | 707/1 | 6959 | L7 |
| L6 | L5 and rows | 134 | L6 |
| L5 | L4 and tables | 174 | L5 |
| L4 | L3 and (object near identif$ or object near id) | 178 | L4 |
| L3 | L2 and (metadata or meta with data) | 427 | L3 |
| L2 | L1 and relation$ near (database or data with base) | 1658 | L2 |
| L1 | (object-oriented or object near oriented) near (database or data with base) | 2742 | L1 |

END OF SEARCH HISTORY

L21: Entry 92 of 94 File: USPT Mar 31, 1998

US-PAT-NO: 5734887
DOCUMENT-IDENTIFIER: US 5734887 A

TITLE: Method and apparatus for logical data access to a physical relational database

DATE-ISSUED: March 31, 1998

INVENTOR-INFORMATION:

| NAME | CITY | STATE | ZIP CODE | COUNTRY |
|---|---|---|---|---|
| Kingberg; Denis G. | Raleigh | NC | | |
| McCubbin; Ellen Margaret | Cary | NC | | |
| Martin; William John | Apex | NC | | |

ASSIGNEE-INFORMATION:

| NAME | CITY | STATE | ZIP CODE | COUNTRY | TYPE CODE |
|---|---|---|---|---|---|
| International Business Machines Corporation | Armonk | NY | | | 02 |

APPL-NO: 08/ 536737 [PALM]
DATE FILED: September 29, 1995

INT-CL: [06] G06 F 17/30

US-CL-ISSUED: 395/604; 395/611
US-CL-CURRENT: 707/4; 707/100

FIELD-OF-SEARCH: 395/604, 395/611

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS



| | PAT-NO | ISSUE-DATE | PATENTEE-NAME | US-CL |
|---|---|---|---|---|
| ☐ | 5206951 | April 1993 | Khoyi et al. | 395/650 |
| ☐ | 5261080 | November 1993 | Khoyi et al. | 395/500 |
| ☐ | 5295222 | March 1994 | Wadhwa et al. | 395/1 |
| ☐ | 5303379 | April 1994 | Khoyi et al. | 395/700 |
| ☐ | 5307484 | April 1994 | Baker et al. | 395/600 |

OTHER PUBLICATIONS

Casey, Logical Data Interface, IBM TDB, vol. 16, No. 4, Sep. 1973 pp. 1203-1207.
Meltzer, Terminology and an Architecture on Data Independence, IBM TDB, vol. 14, No. 12, May 1972, pp. 3709-3712.
IBM TDB, vol. 29, No. 7, Dec. 1986, pp. 2894-2900, Larner, "Access Independent Query Definition in IBM DL/I".
IBM TDB, vol. 26, No. 5, Oct. 1983, pp. 2557-2559, Pullin et al, "Method for Accessing Hierarchical Views of a Binary Relational Database".
IBM TDB, vol. 32, No. 9B, Feb. 1990, pp. 98-102, Ritland, "Call-Type API to SQL/DS with Externally Described Operations".
IBM TDB, vol. 36, No. 7, Jul., '93, pp. 545-546, Anderson et al, "Code Generation for an Object Oriented Applications".
IBM TDB, vol. 28, No. 2, Jul. 1985, p. 561, Chan et al, "Isolating the Application Program from the Physical Database Organization".
Computer, Dec. 1986, pp. 26-36, Mark et al, "Metadata Management" Dec. 1986, Mark et al.
Computer, Dec. 1986, pp. 37-44, Wiederhold, "Views, Objects, and Databases", Gio Wiederhold.
Computer, Jan. 1986, pp. 63-73, Keller, "The Role of Semantics in Translating View Updates", Jan. 1986.
Article by P. Palvia, Memphis State University, Nat'l. Computer Conf.1987, pp. 573-582, "How sensitive is the physical database design? Results . . . ".
Byte, Apr. 1989, pp. 221-233, Orr et al, "Methodology: The Experts Speak".
ACM Trans. on Office Info. Systems, vol. 5, No. 1, Jan. 1987, pp. 48-69, Fishman et al. "Iris: An Object-Oriented Database Management System".
Computer, Dec. 1991, pp. 55-62, Collet et al, "Resource Integration Using a Large Knowledge Base in Carnot".
Dr. Dobb's Journal, Nov. 1994, pp. 36-40 & cont'd. "Database Management in C++".

ART-UNIT: 237

PRIMARY-EXAMINER: Black; Thomas G.

ASSISTANT-EXAMINER: Lintz; Paul K.

ATTY-AGENT-FIRM: Flynn; John D.

ABSTRACT:

Logical Data Access to the physical structure of a relational database is provided for one or more Applications. Applications are developed using the logical entity types and logical entity type attribute names as described in a logical data model. The Applications then use a Logical Data Access Interface to access each of the required physical relational database tables via the Logical Data Access Layer. Applications then use logical entity type and logical entity type attribute names as specified in the Logical Data Model in making Logical Data Requests to the Logical Data Access Layer. The Logical Data Access Layer provides a rich set of functions for allowing an Application to control and manage a database, build and execute database queries and interface with physical database. The Logical Data Access Layer determines which of the physical tables and associated columns are required to satisfy the Application request and then builds one or more database query statements containing the appropriate physical table and column names.

16 Claims, 40 Drawing figures

Previous Doc Next Doc Go to Doc#

L21: Entry 92 of 94 File: USPT Mar 31, 1998

DOCUMENT-IDENTIFIER: US 5734887 A
TITLE: Method and apparatus for logical data access to a physical relational database

Abstract Text (1):
Logical Data Access to the physical structure of a relational database is provided for one or more Applications. Applications are developed using the logical entity types and logical entity type attribute names as described in a logical data model. The Applications then use a Logical Data Access Interface to access each of the required physical relational database tables via the Logical Data Access Layer. Applications then use logical entity type and logical entity type attribute names as specified in the Logical Data Model in making Logical Data Requests to the Logical Data Access Layer. The Logical Data Access Layer provides a rich set of functions for allowing an Application to control and manage a database, build and execute database queries and interface with physical database. The Logical Data Access Layer determines which of the physical tables and associated columns are required to satisfy the Application request and then builds one or more database query statements containing the appropriate physical table and column names.

Brief Summary Text (6):
However, several problems exist with these CASE tools. The Applications written or generated using a CASE tool must still be written against the physical database description (i.e., the physical tables and table attributes). Any time the physical database description changes, Applications utilizing the changed portions must be modified to incorporate the changes. For physical tables and table attributes that are widely used throughout an enterprise system the updating of Applications using particular physical database descriptions can entail a tremendous amount of work in identifying and then properly updating the affected Applications.

Brief Summary Text (7):
Also, in any software system requiring database access, database performance is tuned and enhanced. Tuning may entail a process called normalization, which may involve splitting of tables. Database normalization is a well known technique that is used to decide what relations are needed and what their attributes should be. Normalization can enhance performance, integrity and consistency of the database. Normalization can entail taking a single relation and creating several tables. Different levels of normalization may be utilized. The level of normalization are usually called forms (i.e., first normal form, second normal form, third normal form, fourth normal form, fifth normal form and Boyce-Code Normal form).

Brief Summary Text (8):
Performance may also be enhanced by changing the physical database description to more clearly fit the manner in which the database is being utilized. For instance, if a physical table is being heavily utilized by many applications and very few of the Applications need to write to a particular column then performance may be enhanced by splitting the table into two tables one containing the primary key and column, that requires updating the other table having the primary key and all other columns. Thus, the frequently used new physical table can be read locked by multiple applications thus providing quicker access time. Thus, in analyzing the manner in which the physical tables are utilized, database performance can be

enhanced. In order to enhance performance the physical structure of the database must often be changed.

Brief Summary Text (9):
As has been stated, the process of tuning a database often requires changing the physical tables and table attributes (i.e., the physical description or structure of the database). This in turn requires changes to Applications using the changed tables in order to update the Applications to reflect changes in the physical database tables. Thus, whenever the tables are modified, Applications that utilize the effected tables must be updated to reflect the changes. This is particularly a problem with Application that use the Structured Query Language (SQL) to interface an Application with a relational database.

Brief Summary Text (10):
Using SQL, only physical tables may be updated. An SQL Application must know the physical relationship between the various tables in the database utilized by the Application in order to update or access tables. Application programmers prefer to approach databases from a logical level based on the data needs of the particular Application functions. Although SQL supports schemas and views which permit an application programmer to aggregate tables, these views and schema must be constructed from a knowledge of the physical tables using SQL commands. Thus, although schemas and views permit a higher level of abstraction they still require the application to have information on the physical level (i.e., the physical description or structure of the database).

Brief Summary Text (11):
Another drawback with SQL views is that they cannot be updated directly, the underlying physical tables that are used to create the view must be updated separately when a view record requires updating (i.e., inserting a record or deleting a record). Thus, changes to the underlying physical tables still requires the application to be updated in accordance with the changed physical tables.

Brief Summary Text (12):
Updating an Application is a time consuming error prone process. Even with the advent of CASE tools, changing application code requires an analysis of what needs to be changed, how best to change it, making the changes and regression testing to ensure that the changes actually provide the expected functionality without any unexpected side effects or loss of function. Changes also require the use of computer resources which increases the cost of updating the application. Only after testing and debugging can Applications be brought on line with the updated database tables.

Brief Summary Text (13):
Although the use of CASE tools is the preferred method of creating new database Applications, most of the functionality and data an enterprise requires usually resides in existing applications and databases. Existing Applications sometimes require or expect data in a certain databases to be in certain tables with certain attributes. As databases are re-engineered these physical table structures are changed to fit into the enterprise data model. This re-engineering effort thus entails changes to the existing applications. Often the existing applications are too complex and too costly to re-engineer so they must be updated with the re-engineered data model.

Brief Summary Text (15):
Thus, whenever the physical tables of a database are changed, Applications using the physical tables must also be updated to reflect the database changes.

Brief Summary Text (19):
It is an object of the present invention to provide logical data access to a relational database.

Brief Summary Text (20):
It is a further object of the present invention to provide for making changes to the physical structure of database tables without requiring changes to applications using the tables.

Brief Summary Text (23):
It is still a further object of the present invention to provide the application the ability to update views without knowing the physical table names used in constructing the view.

Brief Summary Text (24):
It is still a further object of the present invention to provide the application the ability to use logical views of a relational database without knowing the join criteria for physical tables used in constructing the views.

Brief Summary Text (31):
In accordance with a preferred embodiment of the present invention is a computer system comprising one or more applications each having a logical data access interface for requesting data access in accordance with a logical data model, said logical data model having a plurality of interrelated logical entity types with each logical entity type having a plurality of logical attributes. The computer system having a relational database management system containing a plurality of physical tables, said physical tables derived from said logical data model, each of said physical tables having a plurality of columns, the database also having a logical to physical data mapping table for mapping each logical entity type and logical attribute pair to a physical table name and a physical column name as stored in the relational database management system and a join table having a join entry for each logical entity type represented by more than one physical table in the relational database management system, each join entry identifying the physical tables to join, the physical columns to join, and the join criteria necessary to form the logical entity type represented by the join entry. The computer system having a logical data access module for receiving a logical database request from a requesting application via the requesting applications's logical data interface, forming one or more database queries having physical table and physical column names using said logical to physical data mapping table and said join criteria table.

Drawing Description Text (5):
FIG. 3 depicts the creation of the data mapping tables.

Drawing Description Text (7):
FIG. 5(a) Logical Customer Table

Drawing Description Text (8):
FIG. 5(b) Logical Item Table

Drawing Description Text (9):
FIG. 5(c) Logical Sales Transaction Table

Drawing Description Text (10):
FIG. 5(d) Logical Payments Table

Drawing Description Text (11):
FIG. 6(a) Physical CUSTOMER Table

Drawing Description Text (12):
FIG. 6(b) Physical INTCUST Table

Drawing Description Text (13):

FIG. 6(c) Physical PAYMENT Table

Drawing Description Text (14):
FIG. 6(d) Physical TTNDR Table

Drawing Description Text (15):
FIG. 6(e) Physical ITEM Table

Drawing Description Text (17):
FIG. 7 Data Mapping Join Table (DMTJOIN).

Drawing Description Text (18):
FIG. 8 Data Mapping Logical to Physical Table DMTLTOP.

Detailed Description Text (4):
In the preferred embodiment the present invention provides Logical Data Access (LDA) to the physical structure of a relational database (RDB). Applications (201,203,205), as shown in FIG. 2 are programmed using the logical entity and attribute names as described in a logical data model. Each Application (201,203,205) may share a logical data model (i.e., an enterprise data model) or have an application specific data model. During new application development the logical data model is driven down to a physical structure or representation of the data as it is stored in a database. The Applications (201,203,205) use a Logical Data Access Interface (LDAI) to access each of the required databases via the Logical Data Access Layer (LDAL) 207. The LDAI uses logical entity and attribute names as specified in the Logical Data Model in making Logical Data Requests(LDR). The LDAL 207 provides a rich set of functions for allowing an Application to control and manage a database, build and execute database queries and interface with physical database. The LDAL 207 determines which of the physical tables, columns and attributes are required and then builds one or more SQL statements, and executes the SQL statement(s) providing any required information to the requesting Application. The LDAL 207 utilizes Data Mapping Tables 209 in order to construct the SQL query from the Applications LDR and the logical entities and attributes used in the LDR. The Data Mapping Tables 209 permit the LDAL 207 to build SQL queries with appropriate physical table and column names and to create appropriate logical entities. The LDAL 207 can then execute the SQL query against the requisite physical database causing the appropriate action such as create, read, update or delete on the database as required by the Application's LDR. The results, if any, of execution of the SQL statement are provided to the requesting Application.

Detailed Description Text (5):
FIG. 3 depicts the construction of a physical database 211 having multiple physical tables 311, 312, 313 as well as a more detailed look at the data mapping tables 209. Note that the logical data model 301 is used to construct the physical database 211. Both the Logical Data Model 301 and the Physical database structure 211 are input to a Data Mapping Utility 303 to construct the Data Mapping Tables 209. The Data Mapping tables 209 are used by the Logical Data Access Layer in SQL statement construction. The Data Mapping Utility 303 is a software system that uses input from the Logical Data Model and the physical database structure to populate the two Data Mapping tables. The Data Mapping tables 209 are shown in more detail as having a Data Mapping Join Table 305 and a Data Mapping Logical to Physical Table 307. Note that most new database applications are developed using a CASE tool. The Data Mapping Utility 303 in the preferred embodiment processes information that is exported from the CASE tool to populate the Data Mapping Tables. All of the components in FIG. 2 and FIG. 3 are described in detail below.

Detailed Description Text (8):
Although there are many techniques for building entity relationship diagrams and creating logical data models and driving the logical data models down to physical database tables, the following entity relationship based technique works well with

the present invention. The first step to creating a logical data model or to creating a physical database design is to develop an entity relationship diagram. In developing the entity relationship diagram the application developer identifies entity relationship types and associated attributes and also the primary key for each entity type. An entity is a thing, e.g. a person or an automobile, a concept, an organization, or an event of interest to the organization, and that which data is to be maintained. An entity type is a classification of an entity satisfying certain criteria. A relationship is an interaction between entities. A relationship type is a classification of relationships based on certain criteria. Usually, nouns in English correspond to entities, while verbs correspond to relationships. In the example shown in FIG. 4 there are four entity types: customer, payments, sales transaction and item. Also shown are several relationship types: makes, are made by, pays for, paid with, contained, are contained in, initiates, and are initiated by. Note that each relationship type has a cardinality. Cardinality is an indication of the relationship between two entities which states whether the relationship is one to one, one to many, zero to one, zero to many. Note that the cardinality is represented on the entity relationship diagram shown in FIG. 4. A line with a single slash across it indicates a one to one relationship between the entities connected by the line. A line with an inverted arrow or crow's feet on the end indicates a one to many relationship between the entities. Any line with a zero on one end of the line indicates that it is possible to have zero to one or zero to many relationships between the data entities. Thus, the markings made on the lines indicate one to one, one to many relationships that define the cardinality of the relationship between the two entities.

Detailed Description Text (10):
The next step is to convert the entity relationship diagram into a conventional file and database structures or database description language. There are many guidelines for performing this step. Typically each entity type is converted into a relation and a relationship type is converted into a standalone relation or consolidated with another relation depending on the cardinality of the relationship. Thus, from the logical data model or the entity relationship diagram it can be shown that all relations are in third normal form, that is all the primary keys of the relationships are derived automatically. The next step in using a traditional entity relationship diagraming technique would be to develop applications based on the file and database structures. Thus, having the physical database designed the developer can use a relational DBMS and write the systems query language program to express the business questions that will be asked of the database. FIG. 12 depicts the output typical of a Data Modeling Tool, and lists the attributes associated with each logical entity. Note that when the logical attribute names are generated in the logical table description as column names (see FIGS. 5a, 5b, 5c, and 5d), any attribute name that is multiple words (for example CUST MARITAL STATUS) has the blanks replaced with underscores (for example CUST.sub.-- MARITAL.sub.-- STATUS). The names used by the applications are the names with underscores in them. The logical level, or logical tables, for entity types shown in FIG. 4 are depicted in FIG. 5. The corresponding physical tables and columns are shown in FIG. 6.

Detailed Description Text (11):
Note that the logical view has four logical entity types (i.e., logical tables) while the physical database structure contains 6 physical tables as depicted in FIG. 6. The Customer entity type is split into two physical tables: CUSTOMER and INTCUST.

Detailed Description Text (12):
Data Mapping Tables

Detailed Description Text (13):
There are two data mapping tables utilized by the present invention: Data Mapping Logical to Physical Table and the Data Mapping Join Table. Each of these tables are

described in detail below. Note also that the preferred embodiment of the present invention uses an Application name variable in the information tags, and puts the application name in the DMTs. The Application Name is not essential to providing Logical Data Access but, provides benefits in that multiple applications can easily be integrated using only one set of DMTs rather than one set per Application. Using the Application name also provides the flexibility of integrating old existing databases with new Applications using a reverse engineered Logical Data Model. Also different applications can use different Logical entity names to refer to the same physical tables. The use of the Application Identifier also provides the DBA with a quick indication of which Applications use which tables. This is helpful in database tuning as well as planning changes to the logical data model. The present invention is thus described with the use of the Application name column.

Detailed Description Text (14):
Data Mapping Logical to Physical Table (DMTLTOP) Physical Representation

Detailed Description Text (15):
The Data Mapping Logical to Physical Table permits the LDAL or Data Access Class (DAC) to map logical entities types and their associated logical attributes into physical database tables and columns. This permits each application developer with ability to name things differently while referring to the same entity. It also provides for greater portability of applications especially when the Applications are ported to use a different relational database definition. The LDAL or DAC uses the Logical to Physical mappings to get physical table names and column names from the logical specification (i.e. logical entity type and logical attributes specified in the LDR). Many databases do not provide for unlimited character length when naming tables and columns. The Logical to Physical mapping table allows the physical names to change (i.e., shortened) without impacting or causing the Application to change. When a physical name is changed all that needs to updated is the entries in the DM tables. A sample DM Logical to Physical Table is shown in FIG. 8 for the Logical Data Model shown in FIG. 5 and the Physical Database shown in FIG. 6. The DM Logical to Physical table is populating using data from the example shown in FIG. 5 and FIG. 6 and with several Applications. Each of the columns are shown in FIG. 8 and are described below.

Detailed Description Text (16):
Column 1--Logical Table Name. This corresponds to a logical table name (i.e., entity name) in the Logical Data Model. In the preferred embodiment, this is a Character field of length 32, and is a primary key to the Data Mapping Table. This field can not be NULL. Note that the Logical Table Name is shown in the LOGTABLE column of the physical table. There are four entity types shown in the DM Logical to Physical table which correspond to the four entity types shown in FIG. 4 and detailed in FIGS. 5(a)-(d) and FIGS. 13-16.

Detailed Description Text (17):
Column 2--Logical Column Name (i.e., entity type attribute or logical attribute). This corresponds to an attribute name in the logical entity type (i.e., the Logical Table Name as described above). In the preferred embodiment, this is a Character field, of length 32, and is a primary key to the Data Mapping Table. This field can not be NULL. The Logical Column Name and Logical Table Name can be used as the primary key to the DM Logical to Physical table. Note that in the preferred embodiment the primary key is a concatenation of the Application Identifier, Logical Table name, Logical Column Name and Physical Table Name.

Detailed Description Text (18):
Column 3--Application Identifier (i.e., Application name). This corresponds to the applications that utilize the logical table/logical column pair. Therefore, if this column is utilized them may be multiple applications having the same logical table logical attribute pair (the physical table and physical column names may differ). One or more applications use the logical tables. In the preferred embodiment, this

is a Character field, of length 18. This field can not be NULL. Use of the Application Identifier provides the benefits discussed above.

Detailed Description Text (19):
Column 4--Physical Table Name. In the preferred embodiment, this corresponds to the physical table name in the RDB. This is a Character field, of length 18, and cannot be NULL. The maximum length of this field is constrained by the maximum length of the RDB table name in the relational database that the application uses.

Detailed Description Text (20):
Column 5--Physical Column Name. In the preferred embodiment, this corresponds to the physical column name in the Physical Table in the RDB. This is a Character field, or length 18, and cannot be NULL. The maximum length of this field is constrained by the maximum length of the RDB column name in the relational database that the application uses.

Detailed Description Text (21):
Data Mapping Join Table (DMTJOIN) Physical Representation

Detailed Description Text (22):
The DMTJOIN Table specifies the physical tables that are used in creating the logical entity type and logical attributes (or SQL views) for the requesting Application. The DMTJOIN Table is used by the Data Access Class (DAC) to determine what tables are to be joined and the conditions under which they are joined. The join criteria is also used to determine how each table is to be updated when the requesting Application requests an update to the logical entity. Thus, the Application is completely isolated from the physical structure of the database. The DAC uses the DMTJOIN table to create SQL views and then to determine the update required of these tables underlying these views when the Application specifies an update in a LDR. A sample DMTJOIN is shown in FIG. 8 for the Logical Data Model shown in FIG. 5 and the Physical Database shown in FIG. 6. Each of the columns are described below.

Detailed Description Text (23):
Column 1--Logical Table Name. In the preferred embodiment, this corresponds to a logical table name in the Logical Data Model. This is a Character field, of length 32, and is a primary key to the Data Mapping Table. In the preferred embodiment the primary key is the Logical Table Name concatenated with the Application Identifier. This field can not be NULL.

Detailed Description Text (24):
Column 2--Application Identifier. In the preferred embodiment, this corresponds to the Applications that utilize the logical table. One or more applications may use the same logical entity type name but have different logical entity type attributes and different join criteria. This is a Character field, of length 18, and is used as part of the primary key in the preferred embodiment. This field can not be NULL.

Detailed Description Text (25):
Column 3--Join Criteria. In the preferred embodiment, this corresponds to the join criteria between physical tables in the RDB that were generated as a result of splitting a logical table in the Logical Data Model into 2 or more physical tables in the physical RDB. Any logical entity type (i.e., logical table) that consists of more then one physical table has an entry (or more then one entry) in the DM Join table. It is a Character field, of maximum length 1000. The Join Criteria identifies the physical tables that the Logical entity type is made from and the criteria under which the tables are joined (i.e., the columns and the criteria under which data in the columns are to be joined). For example:
T1.C1.EQ.T2.C2,T1.C2.NE.T3.C3 would be interpreted as (T1.C1 EQ T2.C2) AND (T1.C2 NE T3.C3) where T1, T2, and T3. are physical table names, and C1, C2, and C3 are

physical column names within those tables, respectively. If multiple columns need to be joined, then the format above can be repeated with a comma separating the join conditions or alternatively multiple entries can be made in the DM join table for each of the conditions to be ANDed together. The comma or multiple DM join entries implies an AND relationship between the join conditions. In FIG. 7 the logical entity type Customer is shown as three entries. Each of the entries are anded together such that the physical table CUSTOMER and INTCUST are joined together where the CUSTOMER.CUSTFIRSTNM=INTCUST.INTCUSTFIRSTNM and CSTOMER.CUSTLASTNM=INTCUST.INTCUSTLASTNM and CUSTOMER.CUSTSSN=INTCUST.INTCUSTSSN. Thus the two tables are joined where the first name, last name and social security number of the respective physical tables are the same.

Detailed Description Text (26):
Application Physical Tables

Detailed Description Text (27):
The physical tables 311 are the tables created from the Logical Data Model 301. These are the actual application tables described in the Logical Data Model, in their physical form. These tables are shown for a sample POS Application in FIG. 6. They are created from the Data

Detailed Description Text (28):
Definition Language (DDL) generated by the CASE tool, or generated by the Application Developer or the Data Modeler. The DDL defines the physical structure of the tables such as the table name, column names and data types for each column. These tables are application or enterprise specific in that their content is dictated by the needs of the Applications that utilize the tables. These tables are populated by Applications or data conversion programs. The application tables reside in the physical relational database 211. Although shown separately in FIG. 3, the physical relational database includes the Data Mapping Tables (DMT) 209 (i.e., Data Mapping Join Table (DMTJOIN) 305 and the Data Mapping Logical to Physical Table 307) along with the Applications Physical Tables 311.

Detailed Description Text (30):
The Application #1 309 is a computer program written like any other program, with the exception that this program uses the Logical Data Access Interface to access data in the relational database. The LDAI permits the Application to build database queries using only the Logical Data Model Specification. In the preferred embodiment the LDAI is a Data Access Class which permits the Application to create/read/write/delete data from a relational database. The Application uses the Logical Entity Types and the Logical Entity Type attributes from the logical data model in the specification of data in the database. The Application thus specifies the data it desires using logical tables and logical table attributes. The Application does not require any knowledge of the physical database structure. The Application is isolated from the physical structure of the database. Changes to the physical structure do not require changes to the Application.

Detailed Description Text (33):
One important Logical Data Request or command available to the Application via the LDAI is the setItemList function. Logical Items are specified with the setItemList method. This method determines the physical table name and column names within those tables. For example, to select logical columns col1, and col2 from logical table A, the setItemList method is called with the following signature:

Detailed Description Text (35):
To build a Dynamic SQL select statement, the application program would then use the buildSelectStmt method, and a physical SQL select statement is returned. All application requests are at the logical level. The Data Access Class returns SQL statements that are at the physical level based on the logical level specification. These returned statements can then be executed using dynamic SQL, and the data is

returned in the dynamic SQL data area (SQLDA). Dynamic SQL and the SQLDA are standard components of any Relational Database system that supports SQL. The DB2 relational data base management system is commercially available from IBM. The Data Access Class may also use the Call Level Interface (CLI) or the ODBC interface or any other dynamic database interface language instead of Dynamic SQL.

Detailed Description Text (38):
The Data Access Class determines which physical table names and physical columns to use in a database query, and the SQL commands required to provide the logical information specified by the Application. The Data Access Class(DAC) is a C++ class that provides Applications access to the physical relational database, and isolates the application from the details of using a RDB. While this DAC is illustrated using C++ the present invention does not require the use of C++ and can be implemented using any programming language.

Detailed Description Text (39):
The DAC implements or carries out the Logical Data Requests or commands or methods from the Application. An Application specifies the logical level data using the following DAC methods setItemList, setSearchCondition, setGroupBy, setHaving, and setOrderBy methods. Each of these methods uses the same basic technique. These methods (object oriented methods look like function calls, but have access to the same data within a class) parse the input string to determine: the logical application name, logical table name, and logical column name. Having obtained the requisite information, a database lookup in the DM Logical to Physical table is used to determine the physical table and column name(s) to use. Each logical table and logical column pair is mapped to one or more physical table and physical column names. The Logical Table name is also used to perform a database lookup against the DM join table. If there are entries in the Join table for the logical table name then the logical table name is composed of multiple physical tables. The join criteria is then obtained for the logical table name if any. The join criteria is then used to construct the logical view.

Detailed Description Text (51):
9. lockTable--lock an RDB table in exclusive mode. This method is invoked by an application when the application wants to prevent other applications from reading or writing a table.

Detailed Description Text (52):
10. fetch--fetch a results row into a SQLDA. This method is invoked by an application when the application is retrieving multiple rows from a database query. This method is invoked after a dynamic SQL statement has been described and prepared, and a cursor created and opened, and is usually used within a loop in the application. When a non-zero return code is received from this method, then the application knows that it has retrieved all rows of data that are applicable to the select statement. This method is only applicable for READING data from the RDB.

Detailed Description Text (60):
18. putVarIntoSearchCondition--places a data value in the search condition section of the SQLDA. This method is similar to the putVarIntoSQLDA method, in that the implementation of the method takes a data value and places it into the SQLDA, based on the logical name. However, since the position in the SQLDA can vary depending on table splits, this method is used to specify data values that are only used in search conditions (SQL WHERE clause).

Detailed Description Text (61):
19. putVarIntoHavingCondition--places a data value in the having condition section of the SQLDA. This method is similar to the putVarIntoSQLDA method, in that the implementation of the method takes a data value and places it into the SQLDA, based on the logical name. However, since the position in the SQLDA can vary depending on table splits, this method is used to specify data values that are only used in

search conditions (SQL WHERE clause).

Detailed Description Text (65):
23. setItemList--set the columns to be used in a query, update, or insert statement. This method is used by an application to specify the logical tables and columns to be used in the SQL query. The implementation of the method requires a lookup of the logical names in the DMT to determine the physical names to use, and join criteria if the tables have been split.

Detailed Description Text (66):
24. setSearchCondition--set the constraints for a SQL statement. This method is invoked by an application when a statement requires a search condition (i.e. a WHERE clause). The application will specify the search condition using the logical names, and the implementation of the method will determine the physical names to use, as well as if the search condition should be broken up into multiple search conditions. This breakup of the search condition may occurs when a logical table has been split into multiple physical tables. The implementation must parse the search condition into subconditions that apply to the physical tables, as a search condition for inserts, updates, and deletes can not span multiple physical tables.

Detailed Description Text (67):
25. setGroupBy--set the group by criteria for a query. This method is invoked by an application when a statement requires data to be grouped in a certain order. In each group of more than one row, all values of each grouping column are equal, and all rows with the same set of values of the grouping columns are in the same group. The application will specify the group columns using the logical names, and the implementation of the method will determine the physical names to use.

Detailed Description Text (68):
26. setHaving--set additional constraints for a SQL statement This method is invoked by an application when a statement requires an intermediate result table that consists of groups of data that meet the constraint specified in the setHaving method. The same processing required for setSearchCondition is invoked in the implementation of this method to determine the having constraint for split logical tables. Logical names are used in the input to this method to specify the constraint, exactly like the setSearchCondition method.

Detailed Description Text (69):
27. setAllDistinct--set duplication constraints for a query. This method is invoked by an application when the application does not want to receive duplicate rows for search conditions. For example, if the application wants to retrieve all the cities that exist in the customer database, the application would build a SELECT DISTINCT CUSTCITY FROM CUSTOMER statement. The application would then not have to worry about duplicate cities being returned.

Detailed Description Text (70):
28. buildSelectStmt--build the SQL select statement. This method is invoked by an application after it has told the data access class which items it wants to read from the database (using the setItemList method with logical names), and specified any constraints on the search of the database (using the setSearchCondition method). This method will return a SQL statement that is built with the physical table and columns names. This statement can then be executed via the executeUsing method of the Data Access Class.

Detailed Description Text (71):
29. buildDeleteStmt--build the SQL delete statement. This method is invoked by an application after it has told the data access class which items it wants to delete from the database (using the setItemList method with logical names), and specified the constraints on the search of the database (using the setSearchCondition method). The implementation of this method will build one or more SQL DELETE

statements, based on whether the logical table is split into multiple physical tables. The application will then ask for the number of tables via the getNumTables method, and get an SQL delete statement for each physical table that exists, via the getDeleteStmt method. The application will then execute the delete statements it got via the executeUsing method of the data access class.

Detailed Description Text (72):
30. buildUpdateStmt--build the SQL update statement. This method is invoked by an application after it has told the data access class which items it wants to update in the database (using the setItemList method with logical names), and specified the constraints on the update of the database (using the setSearchCondition method). The implementation of this method will build one or more SQL UPDATE statements, based on whether the logical table is split into multiple physical tables. The application will then ask for the number of tables via the getNumTables method, and get an SQL update statement for each physical table that exists, via the getUpdateStmt method. The application will then execute the update statements it got via the executeUsing method of the data access class.

Detailed Description Text (73):
31. buildInsertStmt--build the SQL insert statement This method is invoked by an application after it has told the data access class which items it wants to insert in the database (using the setItemList method with logical names), and specified the constraints on the insert of the database (using the setSearchCondition method). The implementation of this method will build one or more SQL INSERT statements, based on whether the logical table is split into multiple physical tables. The application will then ask for the number of tables via the getNumTables method, and get an SQL insert statement for each physical table that exists, via the getInsertStmt method. The application will then execute the insert statements it got via the executeUsing method of the data access class.

Detailed Description Text (74):
32. getDeleteStmt--return the SQL delete statement. The application will invoke this method to get a delete statement from the Data Access Class. The buildDeleteStmt method must be invoked prior to this method. The signature for this method requires a table number. This method will return a physical SQL delete statement that applies for the table specified in the invocation of the method. Prior to invoking this method, the application should invoke the getNumTables method to determine the number of physical tables that are being deleted based on the logical table name that was specified in the setItemList method.

Detailed Description Text (75):
33. getUpdateStmt--return the SQL update statement. The application will invoke this method to get an update statement from the Data Access Class. The buildUpdateStmt method must be invoked prior to this method. The signature for this method requires a table number. This method will return a physical SQL update statement that applies to the table specified in the invocation of the method. Prior to invoking this method, the application should invoke the getNumTables method to determine the number of physical tables that are being updated based on the logical table name that was specified in the setItemList method.

Detailed Description Text (76):
34. getInsertStmt--return the SQL insert statement The application will invoke this method to get an insert statement from the Data Access Class. The buildInsertStmt method must be invoked prior to this method. The signature for this method requires a table number. This method will return a physical SQL insert statement that applies to the table specified in the invocation of the method. Prior to invoking this method, the application should invoke the getNumTables method to determine the number of physical tables that are being updated based on the logical table name that was specified in the setItemList method.

Detailed Description Text (77):
35. getNumTables--return the number of physical tables used in the SQL statements. This method is invoked by an application after the setItemList, setSearchCondition, setGroupBy, and setHaving methods are invoked (not all of these methods need be invoked). After the aforementioned methods are invoked, the Data Access Class knows how many physical tables are affected by the logical table specified in the aforementioned methods. The application needs to know how many physical tables are specified, so the application knows how many physical SQL statements to retrieve from the data access class (via the getDeleteStmt, getUpdateStmt, and getInsertStmt methods). Regardless of the number of physical tables, there is only one Select statement generated, because SQL allows the join of tables in a select statement.

Detailed Description Text (84):
The above listed methods require the Application to use the logical entity and logical entity attribute names in order to obtain logical data access. The implementation of the methods determines the physical tables and columns to use by reading all DM Logical to Physical table entries for the logical entity, attribute, and application name. When the application executes the (for example) buildSelectStmt method, the SQL Select Statement that is built will have the correct physical table(s) and column(s) names in it. In addition, if the logical entity was split into multiple physical tables in the RDB (often done for performance reasons), the join criteria between the physical tables will be added in by the Data Access Layer. This is done by reading all the DMTJOIN entries for the logical entity and application name, and appending the join criteria to any other join criteria specified by the application. The DAC parses each of the arguments supplied in each of the five methods. The DAC then uses each argument and performs a table lookup using the DM Logical to Physical table to obtain the physical tables and column names. The DAC then utilizes the DM Join criteria table to determine if any Logical entity consists of more then one table. If a logical entity is split into one or more physical tables, the DAC gets the physical table names and the join criteria and uses this information to create database statements necessary to create a logical entity type. The logical entity type can then be accessed as if it were a physical table by the Application. If a logical entity type must be updated then the DAC determines the SQL statements that must be generated based on the Join criteria and the request and generates the appropriate update statements for each of the physical tables used in the view as required by the update request. The processing of Logical data requests containing Logical data specifications is shown in FIG. 9.

Detailed Description Text (85):
In FIG. 9 a command is received from an Application by the DAC. In step 801 the DAC parses the command parameters to obtain any Application Identifier, Logical Table Name and Logical Column name tuples embedded in the command parameters. In step 803 the DAC Maps the obtained Application Identifier, Logical Table Name, and Logical Column Name tuple to obtain a Physical Table Name and a Physical Column Name pair. This Logical to Physical mapping is accomplished using the DM Logical to Physical table. In the preferred embodiment this mapping is accomplished using an SQL select statement with the appropriate search terms. For instance, the following select statement can be used:

Detailed Description Text (86):
The Logical to Physical table may contain multiple entries for each of the tuples. In step 805, each obtained Application Identifier and Logical table name pair is used to determine the join criteria if any for the pair. Each Application Identifier and Logical Table Name pair may have more then one entry in the DMTJOIN table as described previously. The entries will be ANDED together if more then one exists. The join criteria obtained from the DMTJOIN table is then parsed to obtain the physical table names that are to be joined and the physical column names in the respective physical table names on which to join the tables and the join conditions. The parsing of the join criteria is shown in step 807. The parsed join

criteria items can then be used to form an SQL statement suitable for creating a view or an SQL join statement. If an Update SQL statement is being built then the parsed join criteria items along with the mapped physical table names and physical column names and command parameters are utilized to construct appropriate SQL statements to update the necessary physical tables. In step 809 the received command is executed and the results, if any, are provided to the requesting Application (i.e., the Application issuing the command). Note that one or more commands may need to be issued in order to build the required SQL statements necessary to accomplish the Applications desired data access. Steps 801 through 807 are performed whenever command parameters contain a logical data specification. Examples of Application code necessary to build and execute queries are shown in the Example of Operation section provided below.

Detailed Description Text (87):
If a Logical Data request specifies that a Logical Entity Type (i.e., logical table) is to be updated then the join criteria for the logical table is utilized to determine which entries in each of the requisite physical tables joined to provide the logical table need to be updated. Updating can result in the insert/deletion/modification of records in one or more of the physical tables joined to form the logical table. The Application does not have to determine which of the physical tables used in the logical table are to updated. The DAC uses the join criteria and the Logical to Physical mapping of the LDR specification to determine which tables are updated and to build the SQL statements required to update the physical tables as required.

Detailed Description Text (88):
The DAC response to Application requests is described in examples provided in the Examples of Operation section. Note that in the preferred embodiment the DMTs are located in the DBMS with the Application tables. In accessing the DMTs (i.e., the DM Logical to Physical Table and the DM Join Table) the DAC may utilize the interface language associated with the DBMS in which the DMTs are stored. Thus, the DAC may use SQL to access required information from the DMTs.

Detailed Description Text (91):
Second, the Data Access Class utilizes each of the Data Mapping Table(s) to generate SQL statements that accept logical table and logical column names as input, and executes SQL with the physical table names and physical column names, as well as any additional join criteria that may apply due to the potential splitting of logical tables into multiple physical tables.

Detailed Description Text (92):
Populating the Data Mapping Tables

Detailed Description Text (93):
As was stated in the Logical Data Modeling section entity-relationship-diagraming techniques are used to capture the essential application data requirements mandated by the needs of an enterprise. Use of entities, the relationships between them, and their corresponding attributes comprise a logical database design evolved in third normal form as a starting point. Once created, the logical data model is used to generate a physical data representation from which a database description may be produced via standard Database Description Language (DDL). The present invention may be implemented by modifying the logical to physical database implementation strategy. As the physical database is created, and logical tables are divided into multiple physical tables for performance reasons, the corresponding join criteria for those physical tables are included in the logical data model for the logical entities (tables) requiring multiple physical tables. This information is utilized to build the Data Mapping Join Table. The physical table names and physical column names must also be captured and are utilized to populate the Data Mapping Logical to Physical table. If an automated tool is utilized, such as ADW, this information may be captured using one of the many descriptive areas in the case tool's

representation of the Logical Data Model. The information captured relates the logical data model to the physical data model and permits the DMTs to be populated. There are of course many techniques for populating the DMTs, one technique is to use the descriptive areas of a CASE tool Logical Data Model, export files from the CASE tool and then process these exported files with a utility that then populates the DMTs. Alternatively, the DBA can populate the tables having access to the logical and physical database descriptions. The DBA could be assisted by a software utility that prompts the DBA for required information. The use of descriptive areas of the Logical Data Model, as it represented in the ADW case tool, and the use of the ADW export files by a Data Mapping Utility that processes the export files to create the DMTs is described in detail below.

Detailed Description Text (94):
Population of the DMTs may be streamlined by embedding Logical to Physical mapping data in the form of tags in the Application's Logical Data Model. These tags are embedded in the descriptive areas of the Logical entity types and their logical attributes. The Logical Data Model is exported with the embedded tag information. These tags are then processed by a utility program, the Data Mapping Utility (DMU), which builds and populates the DMTs. As (or after) the physical database description is completed in the ADW case tool, tags containing the physical table and columns names are placed in the logical attribute description area. The ADW CASE tool is used to generate the export files for the logical design, and also to used to create the DDL necessary for generation of the physical database for the Application(s).

Detailed Description Text (98):
3. As physical tables and columns are defined, they are reflected back to the logical model via the following format:

Detailed Description Text (100):
4. When logical entities are divided into multiple physical tables, the join criteria between those tables must be defined in the following form:

Detailed Description Text (102):
This format specifies the conditions under which columns between tables are joined. If multiple columns need to be joined, then the format above can be repeated with a comma separating the join conditions. The comma implies an AND relationship between the join conditions. For example: T1.C1.EQ.T2.C2,T1.C2.NE.T3.C3 would be interpreted as (T1.C1 EQ T2.C2) AND (T1.C2 NE T3.C3) where T1, T2, and T3. are physical table names, and C1, C2, and C3 are physical column names within those tables, respectively. The application Name that the join applies to is specified by the ApplicationName. The "-s" indicates that this join results from a split table (where one logical entity splits into multiple physical tables). The LogicalEnityName specifies the logical entity that is represented by one or more physical tables.

Detailed Description Text (103):
5. When foreign keys are identified in a table, the target(s) of the foreign key must be specified in the comment area where the foreign key is defined. The format used is:

Detailed Description Text (119):
Each of the above attributes is updated to reflect its intended physical naming convention within the Comment field of the CASE tool for the logical attribute. For the Customer CUST.sub.-- LAST.sub.-- NAME attribute, the structure would be: pos.CUSTOMER.CUSTLASTNM where `pos` represents the application name `CUSTOMER` is the physical table name `CUSTLASTNM` is the physical column name. Each of the remaining attributes would follow similar naming conventions. Each additional application using these attributes would have a dedicated line appended to the Comment field as the one above.

Detailed Description Text (121):
FIG. 3 shows the Data Mapping Utility 303. In the preferred embodiment the Data Mapping Utility (DMU) is a C++ program which uses the export files from the Logical Data Model as input. The DMU analyzes the export files to determine the logical entity types and their logical attributes, as well as their corresponding physical table names and physical column names. In the preferred embodiment the DMU 303 determines the logical entities from the ADW object export file (oi.exp--listing of all data model object types). The DMU correlates the logical entities with the associations export file (ai.exp--listing of all data model associations) to determine the logical attributes associated with the logical entities. Finally, the DMU correlates the logical entities and attributes to the physical table names and physical column names and join criteria via the ADW textual export file (ti.exp--listing of all data model textual information). The physical representation of the logical data model is captured in the text information (i.e., the imbedded tags) associated with description field. (pi.exp--listing of all data model properties)

Detailed Description Text (122):
The DMU 303 takes the logical to physical mappings it has found, and populates two relational database tables. The first RDB table is the Data Mapping Logical to Physical Table (DMTLTOP) 307. The DMTLTOP 307 provides the translation from logical table names from the Logical Data Model 301 to physical table names that are used in the RDB. The second RDB table populated by the DMU 303 is the Data Mapping Join Table (DMTJOIN) 305. The DMTJOIN 305 provides the join criteria between physical tables in the relational database, where the multiple physical tables resulted from the splitting of a logical table in the Logical Data Model into multiple physical tables. While the DMU for this invention is written in C++, it like the other components can be written in any programming language. The DMTLTOP 307 and DMTJOIN 305 are described in more detail above. Note that although each Application could have its own associated DMTs it is preferable that all applications, using a RDB sham the dame DMTs.

Detailed Description Text (124):
The following SQL defines tables used by the DMU program in analyzing the ADW export files. DMU processing consists of reading the export files into temporary database files and performing the actual logical to physical conversion. The following statements create the tables required by the DMU.

Detailed Description Text (125):
The OIRDB table represents the oi.exp export file from ADW. The PIRDB table represents the pi.exp export file from ADW. The AIRDB table represents the ai.exp export file from ADW. The TIRDB table represents the ti.exp export file from ADW. The following paragraphs describe how the Data Mapping Utility pulls this information into the DMTLTOP and DMTJOIN tables used by the Data Access Class.

Detailed Description Text (126):
The DMU utilizes the above tables to process information from the export files. The first step in the process is the parsing of the oi.exp file from ADW. This file contains all the data entities described in the logical data model. The records in the oi.exp file have 3 components. The first is the OID in, which is in columns 1 to 11 of the record. This is a unique identifier for the data entity. It is followed by a comma, and the OID Type. The DMU is interested in OID Type 10007 records. Details of ADW Export record formats can be found in the ADW documentation (ADW/Encyclopedia Management hereby incorporated by reference). Next is a comma followed by a data field in columns 20 thru 32 of the oi.exp record. The data field is the logicalTableName.

Detailed Description Text (127):
The ai.exp export file is examined and the data put in the ADRDB table in the RDB. By placing this export information in the RDB, the DMU can then take advantage of

SQL query capabilities to pull the information needed for the DMTLTOP and DMTJOIN tables. The ai.exp file contains attribute relationships for each logical attribute defined in every logical data entity in the Logical Data Model.

Detailed Description Text (128):
The pi.exp export file from ADW is then processed. This file contains information about the logical columns that relate to the ai.exp file. The only information of interest to the DMU in the pi.exp file are the records that have an object identifier (OLD) type of `30011`. These records are placed into the PIRDB RDB table.

Detailed Description Text (129):
The ti.exp export file from ADW is read. The ti.exp file contains the description areas of the logical attributes. It is this file that also contains the physical column names and join criteria that apply from the physical data tables back to the logical data entities in the model. The tag information is extracted from this file. The DMU searches the ti.exp file for records with an OIDTYPE of `30077`. The tag information extracted is placed into the OIRDB table.

Detailed Description Text (130):
After reading and extracting appropriate data from the ADW export files and placing this information in the various tables (OIRDB, AIRDB, TIRDB, and PIRDB) the tables are joined and the following search is performed.

Detailed Description Text (131):
This search returns records from the database that contain the logical Data entity Name, and the physical Table name, along with the application subject area. Next the data in the text field returned by the select statement above is interrogated to see if it contains any of the tags as discussed previously. The DMU uses the following statement to retrieve multiple lines from the TIRDB table.

Detailed Description Text (132):
These lines are concatenated together to form a complete tag. The need for this processing is dictated by the way ADW stores this information into the ti.exp file. ADW stores a maximum of 73 characters into the text field. The tag commands described above can be greater than 73 characters. When ADW finds the tag with greater than 73 characters, ADW generates an additional record with the same OIDTYPE, and increments the sequence number. The SQL statement above retrieves those lines from the TIRDB table, ordered by their sequence number. The DMU pulls these records together to form one long data string, which contains the tag command. The DMU parses the tag and determines the physical table and column to use, along with any join criteria for that table (used when tables are split). If the TIRDB file contains a foreign key tag (&FK), then there is a pointer to additional table(s) that use this attribute as a foreign key. The DMT is updated for the additional table(s) to indicate that they also contain this attribute.

Detailed Description Text (133):
At this point, the DMU has determined all the logical attributes for a Logical Data Entity, and determined the physical table(s) and column(s) that correspond to the logical data entity. This information is written to the DM Logical to Physical Table. Any join information (in the case of split tables) is also known for the logical data entity at this point in time, and is written to the DM JOIN table.

Detailed Description Text (134):
As the AIRDB, OIRDB, PIRDB, and TIRDB tables are being built, data is committed every 25 transactions. Any errors found in interpreting the ADW export files are written to an output file from the DMU. This output file is useful in not only documenting errors, but often shows errors at the logical data model level where the physical implementation is missing, or partially complete. If no errors are found, the file is empty.

Detailed Description Text (136):
The following examples shows how the logical names from a data model are used in Application. All examples are shown using C++. Each example demonstrates the use of logical data access by specifying logical entities and their associated attributes. The first example shows Application code for application ABC. This code sequence shows the Application statements necessary to use the DAC to setup the SQL statements having the appropriate physical table names and associated physical column names.

Detailed Description Text (137):
The next example takes a selection of data from a database split table. This example shows the selection of all customers from the logical CUSTOMER entity who are married, and then prints out their names. It uses the Logical data model shown in FIG. 5 and the physical tables show in FIG. 6 and the DMT mapping tables are depicted in FIG. 7 and FIG. 8. The use of both DMTs is required since the logical Customer Table consists of two Physical tables (i.e., namely CUSTOMER and INTCUST). The Application code for accomplishing this query using the present invention is shown below.

Detailed Description Text (139):
The third example demonstrates a very important feature of the present invention. This is the ability to update views. The next example shows the statements necessary for updating the logical entity CUSTOMER. The next example demonstrates updating a logical entity. In this example the marital status of a Jane Doe is changed to married. To insert a row into customer may require several SQL statements since the Logical table consist of two physical tables. This example uses the Logical data model shown in FIG. 5 and the physical tables show in FIG. 7 and the DMT mapping tables depicted in FIG. 8. The use of both DMTs is required since the logical Customer Table consists of two Physical tables (i.e., namely CUSTOMER and INTCUST).

Detailed Description Text (140):
Note in the SQL statements below that the Application does not require any infromation on the two physical tables or their join criteria. The SQL statements generated by the DAC in response to the POS application's request to update Customer Logical entity type or logical table is as follows below:

Detailed Description Text (141):
To demonstrate the flexibility of the present invention the Logical entity Customer can be split into another table. For instance the Customer's address could be stored in another Physical table named CUSTADDR having the physical column names ADRCUSSSN, CUSTSTREET, CUST CITY, CUSTPOSTLCD. All that needs to be changed is entries in each of the DMTs. The DM Logical to Physical table needs to be changed to reflect the new table location for this data. The DM Join table also needs to updated to reflect that the Customer entity type or logical table is composed of three physical tables. No application code needs to be modified.

Detailed Description Text (147):
The preferred embodiment of this invention comprises a set of software systems for providing Logical Data Access to a relational database. A single computer system in accordance with the present invention is shown in FIG. 10. FIG. 10 includes a processor 20 connected by means of a system bus 22 to a read only memory (ROM) 24 and memory 38, Memory 38 may consist of any of the following in combination or alone: disk, flash memory, random access memory or read only memory or any other memory technology. Also included in the computer system in FIG. 10 are a display 28 by which the computer presents information to the user, and input devices including a keyboard 26, mouse 34 and other devices that may be attached via input/output port 30. Other input devices such as other pointing devices or voice sensors or image sensors may also be attached. Other pointing devices include tablets, numeric

keypads, touch screens, touch screen overlays, track balls, joy sticks, light pens, thumb wheels, etc. The I/O 30 can be connected to communication lines, disk storage, input devices, output devices, other I/O equipment or other computer systems.

Detailed Description Text (148):
The memory 38 contains several applications 71, 73 and 75 requiring Logical Data Access. The Data Access Class 81 and relational database mangemerit system 83 software systems are also shown in memory. The database management system may be IBM'S DB2. Note that the DM Join Table 91, DM Logical to Physical table 93 and several physical application tables, 94-99 are shown in the DBMS 83. A working memory area 78 is also shown in memory 38. The working memory area 78 can be utilized by any of the elements shown in memory 38. The working memory area 78 may be partitioned amongst software systems and within software systems. The working memory area 78 may be utilized for communication, buffering, temporary storage or storage of data while a software system or function is running. Also shown in memory is the operating system 54.

Detailed Description Text (151):
The present invention allows an application programmer to program at a logical data level, instead of at the physical database level. The physical database structure of tables can be modified without requiring an application to be re-written using the new physical database tables. The invention greatly enhances the ability for Data Administration personnel to tune an existing database design, while isolating the applications which access data from it.

Detailed Description Text (152):
The Present Invention provides several advantages over prior art Database Access techniques. First, Applications can view distributed data as a single logical resource. The Applications do not have to know on which systems the physical tables reside. In fact the Application use of single logical view divorces the Application from any location or network specific information from being included in the Application. Secondly, Database access is simplified through a uniform Application Programming Interface (API) that makes database implementation transparent to applications and application programmers. This translates into reduced development and maintenance costs for Applications. Thirdly, the present invention provides a level of integration for multiple applications to share common data. If an enterprise data model is used in developing all of an enterprises applications and data requirements the Applications can share common data. Fourthly, the present invention permits fine-tuning and/or totally reengineering of the database design for maximum performance with out the need to update or modify applications.

Detailed Description Paragraph Table (1):
TABLE I _____ Data Modeling Definitions TERM DEFINITION _____ ENTITY An entity is a person, place, thing or event that occurs in the real world, about which data is stored. For example, the diagram in FIG. 4 has four entities: customers, sales transactions, payments and items are all entities. Fundamental A fundamental entity exists without reference to any other entities. Associative An associative entity exists primarily to interrelate two or more entities.. Attribute An Attribute entity exists to further describe a fundamental entity. Relationship A relationship describes the association between two entities. From the example in FIG 4, a customer "initiates" a sales transaction. Both customer and sales transaction are entities. Initiates is the relationship between those entities. Cardinality Cardinality is the indication of the relationship between two entities. It reflects the number of instances of the first entity associated with that of the second entity. Attribute An attribute is a type of characteristic of an entity. For example, the entity customer would have such attributes as address, age, and number of family members. Primay Key The attribute or combination of attributes which uniquely identify an entity is known as the primary key for the entity. A Universal

Product Code (UPC) is an example of a primary key identifying the item entity. Alternate Key The alternate key is an alternate choice of atributes which also may uniquely identify an entity instance. For example, an item may be uniquely identified by a description of the item or by the Universal Product Code (UPC) on the item. Foreign Key An attribute or combination of attributes which identifies a relationship between two entities. The foreign key of an entity must be the primay key of a related entity. Foreign keys are used for defining data insert, update and delete rules. Business Rules Business rules dictate the valid values for the attributes of an entity. Examples of business rules corresponding to the FIG. 4 and "a payment must be for a predefined sales transaction" and "a sales transaction may be made by an as yet undefined customer." Business rules preserve the integrity of the database. Normalization Normalization is a process consisting of a set of rules through which data is analysed, decomposed, and simplified to remove redundancy and to ensure consistency. Where appropriate the logical data model represents data in third normal form. A data model or database is in "third normal form" if each non-key attribute depends only on the entity primary key.

---

Detailed Description Paragraph Table (3):

```
_____ __ CREATE
TABLE OIRDB (LOGICALTABLEOID VARCHAR(11)NOT NULL, LOGICALTABLENAME VARCHAR(32)NOT
NULL, PRIMARYKEY(LOGICAL TABLEOID) CREATE TABLE PIRDB (SUBJID CHAR(11) NOT
NULL,PITYPE CHAR(5) NOT NULL, SEQ SMALLINT,PINAME CHAR(32) NOT NULL) CREATE TABLE
AIRDB (AIID CHAR(11) NOT NULL,AITYPE CHAR(5) NOT NULL, FROMID CHA(11) NOT NULL,
TOID CHAR(11) NOT NULL, PRIMARY KEY (AIID)) CREATE TABLE TIRDB (OID CHAR(11) NOT
NULL, OIDTYPE CHAR(5) NOT NULL, SEQENCE SMALLINT NOT NULL, TEXT VARCHAR(73),
PRIMARY KEY (OID, OIDTYPE, SEQUENCE)) CREATE TABLE DMT (LOGTABLECHAR(32) NOT NULL,
LOGCOLUMN CHAR(32) NOT NULL, APPLICATION CHAR(18) NOT NULL, PHYSTABLE CHAR(18) NOT
NULL,PHYSCOLUMN CHAR(18), PRIMARY KEY (LOGTABLE, LOGCOLUMN, APPLICATION,
PHYSTABLE)) CREATE TABLE DMTJOIN (LOGTABLE1 VARCHAR(332) NOT NULL, LOGTABLE2
VARCHAR(32), APPLICATION VARCHAR(18) NOT NULL, JOINCRITERIA VARCHAR(1000) NOT NULL,
PRIMARY KEY (LOGTABLE1, APPLICAITON))
```

---

Detailed Description Paragraph Table (7):

```
_____ __ //
create an instance of the fSQL class fSQL * sqlClassD = new fSQL ( ); long rc=0; //
start the database manager rc=sqlClassD-->sql.sub.-- start ( ); // logon to the
database manager rc=sqlClassD-->sql.sub.-- logon(userid,password); // open the
specific database rc = sqlClassD-->sql.sub.-- open(rdbName); // build the
selectList. The names in the select list are // logical names that the application
is trying to access IString selectList; selectList="POS.CUSTOMER.CUST.sub.--
LAST.sub.-- NAME, POS.CUSTOMER.CUST.sub.-- FIRST.sub.-- NAME"; // tell the data
access class about the logical selection List. At this point the data access class
will // read the DMT table and determine the physical table(s) and column(s) to
access rc=sqlClassD-->setItemList (selectList); // build the where clause - this
constrains the select statement IString whereClauseD;
rc=whereClauseD="POS.CUSTOMER.CUST.sub.-- MARITAL.sub.-- STATUS=`M`"; // build the
where clause. At this point, the data access class will read the DMT table and //
determine the physical table(s) and column(s) to use for the search condition.
sqlClassD-->setSearchCondition(whereClauseD); // allocate the space for the sqlda
sqlda * inputSelectSqldaD; sqlda * outputSelectSqldaD; // the buildSelectStmt
method returns an SQL select statement that contains the PHYSICAL // tables and
columns that correspond to the logical tables and columsn previously specified.
It // also returns the physical SQLDA. IString selectStmtD; sqlClassD--
>buildSelectStmt(&selectStmtD,&inputSelectSqldaD, &outputSelectSqldaD); // At this
point, the select statement is returned to the application with physical // table
and column names. // Note that using dynamic SQL all statements must be Prepared
prior to execution. This is a standard dynamic RDB statement: sqlClassD-->sql.sub.-
- prepare(selectStmtD,"selectStmtD",0); // describe the stmt into the output sqlda
```

where the results of the select stmt will go. All dynamic //select SQL statements must be described. This is a standard dynamic RDB statement. sqlClassD-->sql.sub.-- describe("selectStmtD", outputSelectSqldaD,0); // the select statement may return more than one row from the database. The sql.sub.-- declareCursor is // a standard SQL statement for processing multiple rows returned from the database rc=sqlClassD-->sql.sub.-- declareCursor("selectCursor", "selectStmtD",0); // open the cursor, This is also a standard SQL statement rc=sqlClassD-->sql.sub.-- openCursor ("selectCursor",inputSelectSqldaD,0); // retrieve multiple rows from the database while (rc==0) { // retrieve a row from the database. the Data requested // logical COL1, COL2, and COL3 are in the outputSelectSqldaD rcD=sqlClassD-->sql.sub.-- fetch ("selectCursor", outputSelectSqldaD,0); . if (rcD != 0) { // leave the loop break; } // get the last name and city from the sqlda IString lastName=""; IString firstName=""; SqlClassD-->getVarFromSqlda("POS.CUSTOMER.CUST.sub.-- LAST.sub.-- NAME", &lastName,&outputSelectSqldaD); SqlClassD-->getVarFromSqlda ("POS.CUSTOMER.CUST.sub.-- FIRST.sub.-- NAME", &firstName,&outputSelectSqldaD); // print out the results of the query cout << " name = " << firstName << " " << lastName << endl; } // end while // close the cursor sqlClassD-->sql.sub.-- closeCursor ("selectCursor",0); delete sqlClassD);

---

Detailed Description Paragraph Table (9):

---

create an instance of the fSQL class fSQL * sqlClassD = new fSQL( ); long rc=0; // start the database manager rc=sqlClassD-->sql.sub.-- start( ); // logon to the database manager rc=sqlClassD-->sql.sub.-- logon(userid,password); // open the specific database rc = sqlClassD-->sql.sub.-- open(rdbName); // build the selectList. The names in the select list are logical names that the application is trying to // access IString selectList; selectList="POS.CUSTOMER.CUST.sub.-- MARITAL.sub.-- STATUS"; // tell the data access class about the logical selection List At this point the data access class will // read the DMT table and determine the physical table(s) and column(s) to access rc=sqlClassD-->setItemList (selectList); // build the where clause - this constrains the update statement. The question marks are place // holders that the database manager fills in with values from the sqlda. IString whereClauseD; rc=whereClauseD="(POS.CUSTOMER.CUST.sub.-- LAST.sub.-- NAME=?) AND ( (POS.CUSTOMER.CUST.sub.-- FIST.sub.-- NAME=?)"; // build the where clause. At this point, the data access class // will read the DMT table and determine the physical table(s) // and column(s) to use for the search condition. sqlClassD-->setSearchCondition (whereClauseD); // allocate the space for the sqlda sqlda * inputSqldaD; // put the constrain variables in the dqlda. While these values // are hard coded ehre, they could be input from a user interface, // or passed in as parameters to the program. sqlClassD-->putVarIntoSearchCondition ("POS.CUSTOMER.CUST.sub.-- LAST.sub.- - NAME", "Doe"); sqlClassD-->putVarIntoSearchCondition("POS.CUSTOMER.CUST.sub.-- FIRST.sub. -- NAME", "Jane"); sqlClassD-->putVarIntoSqlda("POS.CUSTOMER.CUST.sub.-- MARITAL.sub.-- STATUS", "M"); // the buildUpdateStmt method builds the physical UPDATE // statement. Since multiple tables may be involved in the update // we need to get the number of tables via the getNumTables method, // and then loop through getting an update statement for each table // that needs to be updated. IString updateStmtD; sqlClassD-->buildUpdateStmt( ); int numTables=sqlClassD-->getNumTables( ); // loop through getting one update statement per table for (int I=0; i<numTables; I++) { sqlClassD-->getUpdateStmt(I,&updateStmtD,&inputSqldaD); // At this point, the update statement that is returned // has the physical table and column names in it, and // is the following: // UPDATE INTCUST SET INTCUST.INTCUSTMARSTATUS=? // WHERE (INTCUST.CUSTLASTNAME=?) AND // (INTCUST.CUSTFIRSTNAME=?) // since INTCUST is the only physical table with // marital status in it, the numTables will // have a value of 1, and this loop will only be // executed once. // prepare the statement. sqlClassD-->sql.sub.-- prepare(updateStmtD,"updateStmtD",0); // execute the statement rc=sqlClassD-->executeUsing("updateStmtD",inputSqldaD); // rc will be the return code from the database manager } // end for delete sqlClasssD

---

Other Reference Publication (4):
IBM TDB, vol. 26, No. 5, Oct. 1983, pp. 2557-2559, Pullin et al, "Method for Accessing Hierarchical Views of a Binary Relational Database".

Other Reference Publication (8):
Computer, Dec. 1986, pp. 26-36, Mark et al, "Metadata Management" Dec. 1986, Mark et al.

Other Reference Publication (13):
ACM Trans. on Office Info. Systems, vol. 5, No. 1, Jan. 1987, pp. 48-69, Fishman et al. "Iris: An Object-Oriented Database Management System".

CLAIMS:

1. A computer system comprising:

one or more applications each having a logical data access interface for requesting data access in accordance with a logical data model, said logical data model having a plurality of interrelated logical entity types with each logical entity type having a plurality of logical attributes;

a relational database management system containing a plurality of physical tables, said physical tables derived from said logical data model, each of said physical tables having a plurality of columns;

a logical to physical data mapping table for mapping each logical entity type and logical attribute pair to a physical table name and a physical column name as stored in the relational database management system;

a join table having a join entry for each logical entity type represented by more than one physical table in the relational database management system, each join entry identifying the physical tables to join, the physical columns to join, and the join criteria necessary to form the logical entity type represented by the join entry;

a logical data access module for receiving a logical database request from a requesting application via the requesting applications's logical data interface, forming one or more database queries having physical table and physical column names using said logical to physical data mapping table and said join criteria table.

6. The computer system according to claim 5 wherein the join table and the logical to physical mapping table are stored with the relational database management system.

7. The computer system according to claim 6 wherein the logical data request also includes an application identifier and the logical to physical mapping table and the join table each include an application identifier column.

8. A method of accessing data in a relational database comprising:

receiving a logical data request that specifies logical entity types and logical entity type attribute in accordance with a logical data model;

parsing said logical request to obtain one or more logical entity type and logical attribute tuples;

mapping each logical entity type and logical attribute tuple to a physical table and a physical table column;

obtaining the join criteria for each logical entity type represented by more then one physical tables, the join criteria associated with the logical entity type specifying how to join the physical tables representing the logical entity type to form the logical entity type table;

building one or more dynamic SQL statements in accordance with the logical data request that may be executed against the physical tables in the relational database.

10. The method of claim 8 wherein the join criteria obtained is parsed to determine the physical tables and associated physical columns to use in join statements and the join conditions on which to join the columns.

11. An article of manufacture comprising a computer useable medium having a computer readable program embodied in said medium, wherein the computer readable program when executed on a computer causes the computer to:

receive a logical data request that specifies logical entity types and logical entity type attribute in accordance with a logical data model;

parse said logical request to obtain one or more logical entity type and logical attribute tuples;

map each logical entity type and logical attribute tuple to a physical table and a physical table column;

obtain the join criteria for each logical entity type represented by more then one physical tables, the join criteria associated with the logical entity type specifying how to join the physical tables representing the logical entity type to form the logical entity type table;

build one or more dynamic SQL statements in accordance with the logical data request that may be executed against the physical tables in the relational database.

16. A computer system comprising:

one or more applications each having a logical data access interface for requesting data access in accordance with a logical data model, said logical data model having a plurality of logical tables with each logical table having a plurality of logical attributes;

a relational database management system containing a plurality of physical tables, said physical tables derived from said logical tables, each of said physical tables having a plurality of columns;

a logical to physical data mapping table for mapping each logical table and logical table attribute pair to a physical table name and a physical column name as stored in the relational database management system;

a join table having a join entry for each logical table represented by more than one physical table in the relational database management system, each join entry identifying the physical tables to join, the physical columns to join, and the join criteria necessary to form the logical table represented by the join entry;

a logical data access module for receiving a logical database request from a requesting application via the requesting applications's logical data interface, forming one or more database queries having physical table and physical column names using said logical to physical data mapping table and said join criteria

table.

Previous Doc Next Doc Go to Doc#

L20: Entry 113 of 124 File: USPT Nov 30, 1999

US-PAT-NO: 5995973
DOCUMENT-IDENTIFIER: US 5995973 A

TITLE: Storing relationship tables identifying object relationships

DATE-ISSUED: November 30, 1999

INVENTOR-INFORMATION:

| NAME | CITY | STATE | ZIP CODE | COUNTRY |
|---|---|---|---|---|
| Daudenarde; Jean-Jacques P. | San Jose | CA | | |

ASSIGNEE-INFORMATION:

| NAME | CITY | STATE | ZIP CODE | COUNTRY | TYPE CODE |
|---|---|---|---|---|---|
| International Business Machines Corporation | Armonk | NY | | | 02 |

APPL-NO: 08/ 921198 [PALM]
DATE FILED: August 29, 1997

PARENT-CASE:
CROSS-REFERENCE TO RELATED APPLICATION This application is related to the following co-pending and commonly-assigned patent application: Application Ser. No. 08/921,196, entitled "SCALABLE DATA WAREHOUSE ARCHITECTURE," filed on same date herewith, by Linnette M. Bakow, et al., attorney's docket number ST9-97-075, which is incorporated by reference herein.

INT-CL: [06] G04 F 17/30

US-CL-ISSUED: 707/103; 707/1, 707/3, 707/100, 707/102
US-CL-CURRENT: 707/103R; 707/1, 707/100, 707/102, 707/3

FIELD-OF-SEARCH: 707/2, 707/3, 707/4, 707/100, 707/102, 707/103, 364/282.1, 364/283.4, 395/680, 395/683

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS



| | PAT-NO | ISSUE-DATE | PATENTEE-NAME | US-CL |
|---|---|---|---|---|
| ☐ | 5263165 | November 1993 | Janis | 711/163 |
| ☐ | 5315709 | May 1994 | Alston, Jr. et al. | 707/6 |
| ☐ | 5379419 | January 1995 | Heffernan et al. | 707/3 |

| | | | | |
|---|---|---|---|---|
| ☐ | 5526492 | June 1996 | Ishida | 395/200.09 |
| ☐ | 5561769 | October 1996 | Kumar et al. | 395/200.05 |
| ☐ | 5583997 | December 1996 | Hart | 395/200.15 |
| ☐ | 5592626 | January 1997 | Papadimitriou et al. | 395/200.09 |
| ☐ | 5596744 | January 1997 | Dao et al. | 707/10 |
| ☐ | 5608720 | March 1997 | Biegel et al. | 370/249 |
| ☐ | 5687362 | November 1997 | Bhargava et al. | 707/2 |
| ☐ | 5729739 | March 1998 | Cantin et al. | 707/103 |
| ☐ | 5765159 | June 1998 | Srinivansan | 707/102 |
| ☐ | 5797136 | August 1998 | Boyer et al. | 707/2 |
| ☐ | 5850544 | December 1998 | Parvathaney et al. | 707/101 |
| ☐ | 5873093 | February 1999 | Williamson et al. | 707/103 |
| ☐ | 5878411 | March 1999 | Burroughs et al. | 707/4 |
| ☐ | 5893913 | April 1999 | Brodsky et al. | 707/201 |

OTHER PUBLICATIONS

IBM Technical Disclosure Bulletin, vol. 38, No. 01, Jan. 1995, pp. 607-608.
IBM Internet article, http://www.accu-info.com/ibm18-01.htm, "Great New Ads", NewsBooth #18, Oct. 3, 1996 (entire document).
Brower, D. et al., Internet article, http://globecom.net/ietf/rfc/rfc1697.shtml, "RFC1697: Relational Database Management System (RDBMS) Management Information Base (MIB) using SMIv2", pp. 1-36 no date.
National Geophysical Data Center/WDC-A For Solid Earth Geophysics, Internet article, http://julius.ngdc.noaa.gov/seg/prosol.html, "NGDC Hierarchical Data & Metadata Server", Jan. 1997 (entire document).
Internet article, http://skydive.ncsa.ui...present/wm16/fd15.html, "URL/Metadata Database-Georgia Tech" (entire document).
Internet article, http://skydive.ncsa.ui...present/wm16/fd16.html, "Bidirectional Links for the Web" (entire document).
Berkeley Multimedia Research Center, Internet article, http://www.bmrc.berkel...ers/VodsArch94.html#24, "A Distributed Hierarchial Storage Manager for a Video-on-Demand System", Storage and Retrieval for Image and Video Databases II, IS&T/SPIE Symp. on Elec. Imaging Sci. & Tech., San Jose, CA, Feb. 1994, pp. 1-16.

ART-UNIT: 271

PRIMARY-EXAMINER: Lintz; Paul R.

ASSISTANT-EXAMINER: Colbert; Ella

ATTY-AGENT-FIRM: Pretty, Schroeder & Poplawski, P.C.

ABSTRACT:

A method, apparatus, and article of manufacture for maintaining object relationships. A query is executed in a computer to retrieve data from a database stored on a data storage device. Initially, one or more relationship objects are created, wherein each relationship object identifies a source object and a related

target object. Next, when accessing a source object, related target objects are determined using the relationship object. When accessing a target object, related source objects are determined using the relationship object.

69 Claims, 5 Drawing figures

Previous Doc Next Doc Go to Doc#

L20: Entry 115 of 124 File: USPT Sep 21, 1999

US-PAT-NO: 5956730
DOCUMENT-IDENTIFIER: US 5956730 A

TITLE: Legacy subclassing

DATE-ISSUED: September 21, 1999

INVENTOR-INFORMATION:

| NAME | CITY | STATE | ZIP CODE | COUNTRY |
|---|---|---|---|---|
| Burroughs; Tracy Kim | Byron | MN | | |
| Lee; Wilson D. | Rochester | MN | | |
| Rogers; Cynthia Ann | Rochester | MN | | |
| Zaborowski; Laura Jane | Winona | MN | | |

ASSIGNEE-INFORMATION:

| NAME | CITY | STATE | ZIP CODE | COUNTRY | TYPE CODE |
|---|---|---|---|---|---|
| International Business Machines Corporation | Armonk | NY | | | 02 |

APPL-NO: 08/ 912020 [PALM]
DATE FILED: August 15, 1997

INT-CL: [06] G06 F 17/30

US-CL-ISSUED: 707/104; 707/103, 707/102, 395/683
US-CL-CURRENT: 707/104.1; 707/102, 719/315

FIELD-OF-SEARCH: 707/10, 707/103, 707/104, 707/102, 395/683

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS



| | PAT-NO | ISSUE-DATE | PATENTEE-NAME | US-CL |
|---|---|---|---|---|
| ☐ | 5542078 | July 1996 | Martel et al. | 395/600 |
| ☐ | 5604892 | February 1997 | Nuttall et al. | 395/500 |
| ☐ | 5613099 | March 1997 | Eriickson et al. | 395/500 |
| ☐ | 5627979 | May 1997 | Chang et al. | 395/335 |
| ☐ | 5692183 | November 1997 | Hapner et al. | 395/614 |
| | 5694598 | December 1997 | Durand et al. | 395/614 |

| | | | | |
|---|---|---|---|---|
| ☐ | | | | |
| ☐ | 5729739 | March 1998 | Cantin et al. | 395/614 |
| ☐ | 5737597 | April 1998 | Blackman et al. | 395/613 |
| ☐ | 5799310 | August 1998 | Anderson et al. | 707/102 |
| ☐ | 5832496 | November 1998 | Anand et al. | 707/102 |
| ☐ | 5870746 | February 1999 | Knutson et al. | 707/101 |
| ☐ | 5870749 | February 1999 | Adusumilli | 707/101 |

OTHER PUBLICATIONS

Enabling the Integration of Object Applications with Relational Databases, by Arthur Keller, Ph.D., Richard Jensen, and Shailesh Agrawal, Ph.D., Persistence Software, Mar. 24, 1997
Inc.,www.persistence.com/pesr...ce/pagetwo.pages/technoview.html, pp. 1-9.
Java Relational Binding Delivers Transparent Java Access to Relational Databases, http://www.O2tech.com, O.sub.2 Technology, Oct. 7, 1996.
XDB's Jet Series (Java Enterprise Tools), XDB Systems, http://www.xdb.com/jet, Mar. 24, 1997.
Jetstore, XDB Systems, http://www.xdb.com/jet/store, p. 1, Mar. 24, 1997.
Jetassist Instant Database Applet Development for Java, XDB Systems, http://www.xdb.com/jet/assist, pp. 1-2, Mar. 24, 1997.
Jetconnect Universal Database Access for Java, XDB Systems, http://www.xdb.com/jet/connect, pp. 1-2, Mar. 24, 1997.
ObjectStore DBconnect, ObjectStore Database for the Web http://www.odi.com/products/apc/ap.sub.-- connect.html, pp. 1-5, Mar. 24, 1997.
Welcome to O.sub.2 Technology Database Solutions for Object Developers, http://www.O2tech.fr, pp. 1-2, Mar. 24, 1997.

ART-UNIT: 276

PRIMARY-EXAMINER: Kulik; Paul V.

ASSISTANT-EXAMINER: Wallace, Jr.; Michael J.

ATTY-AGENT-FIRM: Maxwell; Lawrence D.

ABSTRACT:

A method and system for mapping between relational schema and object schema, wherein the relational schema includes a table having a tiebreaker column. In accordance with the present invention, an object-oriented application program may instantiate a persistent dependent object with one of two or more specific instances that is selected in response to the value of a data element in a tiebreaker column. Alternatively, an object-oriented application program may instantiate a persistent entity object in accordance with one of two or more entity classes that is selected in response to the value of a data element in a tiebreaker column.

14 Claims, 9 Drawing figures

Previous Doc Next Doc Go to Doc#

L20: Entry 119 of 124 File: USPT Feb 16, 1999

US-PAT-NO: 5872973
DOCUMENT-IDENTIFIER: US 5872973 A

TITLE: Method for managing dynamic relations between objects in dynamic object-oriented languages

DATE-ISSUED: February 16, 1999

INVENTOR-INFORMATION:

| NAME | CITY | STATE | ZIP CODE | COUNTRY |
|---|---|---|---|---|
| Mitchell; David C. | South Orem | UT | | |
| Anderson; Kelly L. | Provo | UT | | |
| Osman; Andrew V. | Provo | UT | | |
| Mitchell; Dale K. | Provo | UT | | |

ASSIGNEE-INFORMATION:

| NAME | CITY | STATE | ZIP CODE | COUNTRY | TYPE CODE |
|---|---|---|---|---|---|
| Viewsoft, Inc. | Provo | UT | | | 02 |

APPL-NO: 08/ 548536 [PALM]
DATE FILED: October 26, 1995

INT-CL: [06] G06 F 9/44

US-CL-ISSUED: 395/685; 395/683, 395/702
US-CL-CURRENT: 719/332; 717/108, 717/116, 719/317, 719/330

FIELD-OF-SEARCH: 395/685, 395/710, 395/614, 395/683, 395/701, 395/702, 395/708, 364/284.4, 707/103

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

Search Selected Search ALL Clear

| | PAT-NO | ISSUE-DATE | PATENTEE-NAME | US-CL |
|---|---|---|---|---|
| ☐ | 5327562 | July 1994 | Adcock | 395/700 |
| ☐ | 5339430 | August 1994 | Lundin et al. | 395/700 |
| ☐ | 5359721 | October 1994 | Kempf et al. | 395/425 |
| ☐ | 5367633 | November 1994 | Matheny et al. | 395/164 |
| ☐ | 5369766 | November 1994 | Nakano et al. | 395/700 |

| | | | | |
|---|---|---|---|---|
| ☐ | 5371891 | December 1994 | Gray et al. | 395/700 |
| ☐ | 5418964 | May 1995 | Lonner et al. | 395/700 |
| ☐ | 5423841 | June 1995 | Bunke et al. | 395/700 |
| ☐ | 5437025 | July 1995 | Bale et al. | 395/600 |
| ☐ | 5485671 | January 1996 | Stutz et al. | 395/700 |
| ☐ | 5515536 | May 1996 | Corbett et al. | 395/700 |
| ☐ | 5619638 | April 1997 | Duggan et al. | 395/703 |

OTHER PUBLICATIONS

Huy Hguyen et al., OODDM: An Object-Oriented Database Design Model, Apr., 1990, see page 339, right-hand column, line 35; and page 341, left-hand column, line 2.
C. Popien et al., A Concept For An ODP Service Management, IEEE, 1994, see pages 888-897.
Cheng et al, "On The Performance Issues of Object-Based Buffering", Proceedings of the First International Conference on Parallel and Distributed Information Systems, p. 30-37, 4 Dec. 1991.

ART-UNIT: 274

PRIMARY-EXAMINER: Trammell; James P.

ASSISTANT-EXAMINER: Chaki; Kakali

ATTY-AGENT-FIRM: Wilson Sonsini Goodrich & Rosati

ABSTRACT:

A method and system for creating named relations between classes in a dynamic object-oriented programming environment via mappers is disclosed. The mapping objects dynamically bind to the class interfaces of the classes being related. These connections between classes are defined within a visual environment. The relationships can be programmatically attached by name to object instances during program execution. Because these relationships are stored in a resource and are dynamically bound by name to the objects, they can be created and modified without requiring the source code of the objects being associated to be changed. This eliminates hard coded dependencies between objects that impede reuse of the objects in other contexts. The invention requires and takes full advantage of, meta-data, full dynamic binding and probing support in the objects being connected with the invention.

33 Claims, 4 Drawing figures

Previous Doc Next Doc Go to Doc#

L20: Entry 120 of 124 File: USPT Nov 17, 1998

US-PAT-NO: 5838965
DOCUMENT-IDENTIFIER: US 5838965 A

TITLE: Object oriented database management system

DATE-ISSUED: November 17, 1998

INVENTOR-INFORMATION:

| NAME | CITY | STATE | ZIP CODE | COUNTRY |
|---|---|---|---|---|
| Kavanagh; Thomas S. | Boulder | CO | | |
| Beall; Christopher W. | Boulder | CO | | |
| Heiny; William C. | Arvada | CO | | |
| Motycka; John D. | Evergreen | CO | | |
| Pendleton; Samuel S. | Louisville | CO | | |
| Smallwood; Thomas D. | Lafayette | CO | | |
| Terpening; Brooke E. | Golden | CO | | |
| Traut; Kenneth A. | Boulder | CO | | |

ASSIGNEE-INFORMATION:

| NAME | CITY | STATE | ZIP CODE | COUNTRY | TYPE CODE |
|---|---|---|---|---|---|
| CADIS, Inc. | Boulder | CO | | | 02 |

APPL-NO: 08/ 339481 [PALM]
DATE FILED: November 10, 1994

INT-CL: [06] G06 F 17/30

US-CL-ISSUED: 395/614
US-CL-CURRENT: 707/103R

FIELD-OF-SEARCH: 395/614, 395/600

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS



| | PAT-NO | ISSUE-DATE | PATENTEE-NAME | US-CL |
|---|---|---|---|---|
| ☐ | 3343133 | September 1967 | Dirks | 340/172.5 |
| ☐ | 4318184 | March 1982 | Millett et al. | 364/900 |
| ☐ | 4879648 | November 1989 | Cochran et al. | 364/300 |

| | | | | |
|---|---|---|---|---|
| ☐ | 4887206 | December 1989 | Natarajan | 364/401 |
| ☐ | 4918602 | April 1990 | Bone et al. | 364/401 |
| ☐ | 4930071 | May 1990 | Tou et al. | 364/300 |
| ☐ | 4984155 | January 1991 | Geier et al. | 364/401 |
| ☐ | 5021992 | June 1991 | Kondo | 364/900 |
| ☐ | 5109337 | April 1992 | Ferriter et al. | 364/401 |
| ☐ | 5133075 | July 1992 | Risch | 395/800 |
| ☐ | 5146404 | September 1992 | Calloway et al. | 364/401 |
| ☐ | 5191534 | March 1993 | Orr et al. | 364/468 |
| ☐ | 5206949 | April 1993 | Cochran et al. | 395/600 |
| ☐ | 5210868 | May 1993 | Shimada et al. | 395/600 |
| ☐ | 5241624 | August 1993 | Torres | 395/129 |
| ☐ | 5257365 | October 1993 | Powers et al. | 395/600 |
| ☐ | 5260866 | November 1993 | Lisinki et al. | 364/401 |
| ☐ | 5283865 | February 1994 | Johnson | 395/161 |
| ☐ | 5291583 | March 1994 | Bapat | 395/500 |
| ☐ | 5335346 | August 1994 | Fabbio | 395/600 |
| ☐ | 5379430 | January 1995 | Nguyen | 395/603 |
| ☐ | 5418942 | May 1995 | Krawchuck et al. | 395/600 |
| ☐ | 5418961 | May 1995 | Segal et al. | 395/700 |
| ☐ | 5423038 | June 1995 | Davis | 395/650 |
| ☐ | 5434791 | July 1995 | Koko et al. | 364/468.03 |
| ☐ | 5446842 | August 1995 | Schaeffer et al. | 395/200.01 |
| ☐ | 5448726 | September 1995 | Cramsie et al. | 395/600 |
| ☐ | 5542078 | July 1996 | Martel et al. | 395/600 |
| ☐ | 5546577 | August 1996 | Marlin et al. | 395/614 |

OTHER PUBLICATIONS

International Search Report, International Application No. PCT/US95/15028 dated Apr. 22, 1996.

ART-UNIT: 237

PRIMARY-EXAMINER: Amsbury; Wayne

ATTY-AGENT-FIRM: Baker & Botts, L.L.P.

ABSTRACT:

A database management system is disclosed having an object oriented representation of information describing characteristics of instances organized in a hierarchical structure that may be logically represented as a tree structure. The hierarchical

structure includes a parent-child/class-subclass structure. The internal representation of an instance is dependent upon information that is locally available from a class to which that instance belongs plus inherited attributes from a parent class. A class is represented as a class object having a handle. The class object has a parent handle associated with it that identifies the parent class of the class object. The class object has a subclass list associated with it that identifies the handles of the classes that are subclasses of the class object. The class object has an attribute list associated with it that includes a list of handles which may be used to identify the attributes of the class object. A class object also includes a subtree instance count which represents the total number of instances that belong to that class object plus the total number of instances that are present in all of the descendants of the class object, i.e., the total number of instances that are present in that branch of the hierarchical tree structure. A graphical user interface is provided in which the hierarchical tree structure is displayed in a window showing classification structure, and a class or instance can be selected by clicking on the graphical representation of the class in the window showing classification structure. Attributes of the selected class or instance are simultaneously displayed in a window showing attribute information, which includes fields to the right of each attribute in which search criteria for that attribute may be entered. The subtree instance count is also displayed simultaneously to provide feedback to the user as to how many instances satisfy the current query, in order to facilitate guided iterative queries of the database.

47 Claims, 204 Drawing figures

Previous Doc Next Doc Go to Doc#